9/85
Thinking Machines
Propaganda

Digitized by the Internet Archive
in 2026 with funding from
Brewster Kahle

https://archive.org/details/thinkingmachines00unse_8

Thinking Machines Corporation

Thinking Machines Corporation

Thinking Machines Corporation is a privately held company devoted to the application of parallel processing and artificial intelligence technologies. The company has assembled a team that combines talented scientists and engineers with senior management that has significant achievements in business, technical innovation, and customer service.

Sheryl Handler
President

Ph.D., MIT.

Sheryl Handler was educated at Case Western Reserve, Harvard and MIT, where she was a Collamore-Rogers Scholar.

Prior to founding Thinking Machines Corporation, Sheryl was President for 12 years of PACE/CRUX, a domestic and international economic development firm. Clients ranged from biotechnology and telecommunication companies to the World Bank, the U.S. State Department, the U.N. and numerous other agencies and companies. Her experience in working across scientific, engineering and business disciplines provided the basis for assembling the world-class team behind Thinking Machines.

Richard J. Clayton
Vice President for Product Development

M.S.E.E., MIT.

During his 20 year career at Digital Equipment Corp., Dick was Vice President of Computer Systems Development and Vice President of Advanced Manufacturing Technology. His responsibilities included management of the original VAX design effort. Prior to that he was Group Product Line Manager of Medium Systems.

Dick is responsible for engineering and commercial introduction of the Connection Machine™ product. He has used his experience in bringing other major computer families to market to focus the Connection Machine engineering effort on sound design, ease of manufacturing, and fast fault diagnosis.

Marvin Denicoff
Chief of Project Development

M.A., Mexico City College, Mexico.

For over 30 years, Marvin was the architect of U.S. government programs in artificial intelligence and related research. As Director of the Computer Science program at the Office of Naval Research, he directed DOD's largest basic research program in artificial intelligence, man-machine systems, and advanced software. His many honors and awards include the DOD/Navy Distinguished Civilian Service Award.

As Chief of Project Development and a founder, Marvin structures joint projects between Thinking Machines and corporations, government agencies, and universities.

W. Daniel Hillis
Founding Scientist

Ph.D., MIT.

Danny is an acknowledged industry leader in massively parallel systems design. He has made important contributions to artificial intelligence applications in the fields of common sense reasoning and robotics, as well as in the field of systems architecture. He is a former Hertz fellow and author of the book *The Connection Machine*.

Danny is the architect of the Connection Machine system. The design of the hardware is a direct outgrowth of his pioneering work in parallel algorithms and software.

Mirza Mehdi
Vice President for Corporate Development

M.B.A., Georgia State University.

Mirza's experience and accomplishments span the range from major multi-national firms to vigorous young companies. After honing his analytical skills as a Certified Public Accountant with Arthur Andersen & Co., he managed the Business Planning Department of the International Division of Baxter-Travenol Laboratories, where his responsibilities included strategic planning and evaluation of potential acquisitions. Prior to joining Thinking Machines, Mirza was the Director of Finance and Business Development for Genetics Institute, Inc., a major biotechnology firm.

At Thinking Machines, Mirza provides expertise in strategic planning, project evaluation and financial management, and an overview that bridges the scientific and business activities of the company.

Marvin Minsky
Founding Scientist

Donner Professor of Science
MIT

Ph.D., Princeton.

Marvin is a father of the artificial intelligence field and one of its most influential leaders. His work has emphasized approaches to problems of symbolic description, knowledge representation, semantics, machine perception and learning and, recently, in psychological and physiological theories of imagery, memory, and new computational structures. Also an experienced engineer, Marvin was one of the most influential initiators of the modern field of intelligence-based mechanical robots. Among his many honors and awards, he is a Member of the National Academy of Sciences, a Fellow of the American Academy of Arts and Sciences, and the winner of the Turing Award of the Association for Computing Machinery.

At Thinking Machines, Marvin serves as scientific advisor for the company's artificial intelligence projects.

Howard L. Resnikoff
Vice President and Director of Research

Ph.D., University of California, Berkeley.

Howard's background and achievements span the government and university communities. He was the founding Director of the Division of Information Science and Technology at the National Science Foundation. Prior to that he was Chairman of the Mathematics Department at the University of California, Irvine. While at Irvine he was chairman of the California Statewide Committee on Research. Most recently he was Associate Vice President for Information Services and Technology at Harvard. Howard has published in pure mathematics, natural language processing, and vision. Among his many honors is the U.S. Senior Scientist Award from the Alexander von Humboldt Foundation of the Federal Republic of Germany.

Howard, a founder of Thinking Machines, directs research and product development in VLSI design automation systems, natural language understanding technology for data base access, machine vision, and the development of parallel algorithms for the Connection Machine system.

James Bailey
Director of Marketing

B.A., Brown University.

During his 16 year career at Digital Equipment Corporation, Jim managed the New Products Marketing department, which introduced major new corporate products, including the VAX. He has also held positions of corporate manager of pricing, competitive analysis, and market research. He is a member of Phi Beta Kappa.

Jim is responsible for introducing Thinking Machines products and technology to customers, and for establishing joint research programs where appropriate to assure a successful bridge between our technology and the customer application.

Rolf-Dieter Fiebrich
Senior Scientist

Dr.rer.nat (Ph.D.) in Computer Science,
Technical University of Munich.

During Rolf's career at IBM he was a Research Staff Member of the Thomas J. Watson Research Center, where he focused on research programs for methods of VLSI design. Prior to that he was Assistant Professor of Computer Science at Ludwig-Maximilians-University in Munich. He has published many articles in the field of VLSI design.

Rolf manages the company's VLSI design automation development. The set of tools he has specified provide a new level of performance and integration. These tools allow tighter collaboration between systems architects and chip designers. Rolf is responsible for the custom VLSI program that supports the Connection Machine engineering effort. He also manages the development of VLSI design tools that exploit the power of the Connection Machine system.

George Robertson
Senior Scientist

M.S., Carnegie-Mellon University.

Before joining Thinking Machines, George was Senior Scientist at Bolt, Beranek, and Newman, and Research Computer Scientist in Artificial Intelligence at Carnegie-Mellon. He has done research in distributed systems, multiprocessor systems, programming languages, and user interfaces. He was the principal designer of the large scale ZOG decision support system for the USS Carl Vinson nuclear aircraft carrier.

At Thinking Machines, George is involved in a wide range of research projects, including the design of the TMC Indexer interface and advanced testing strategies for the Connection Machine system.

Guy L. Steele Jr.
Senior Scientist

Ph.D., MIT.

Prior to joining Thinking Machines, Guy was Assistant Professor of Computer Science at Carnegie-Mellon University, where he engaged in research in VLSI design, computer architectures, and high-level languages. Guy is the author of *Common Lisp: The Language*, the standard text on the subject. He is also a co-author (with Samuel P. Harbison) of *C: A Reference Manual*. He was Program Chairman for the 1984 ACM Symposium on LISP and Functional Programming, and has served on the program committees of other ACM symposia.

Guy leads the systems software effort for the Connection Machine system. He has defined the specification of the CM Lisp™ and C*™ languages. He is also managing the development of lower-level languages and microcode for the system.

David L. Waltz
Senior Scientist

Ph.D., MIT.

Dave joined Thinking Machines from the University of Illinois, Urbana, where he was Professor of Electrical Engineering and Research Professor at the Coordinated Science Laboratory. Prior to that he was a post-doctoral researcher at the Artificial Intelligence Lab at MIT. He was the Editor for AI of the *Communications of the ACM* and Executive editor of *Cognitive Science*. He has published widely in the field of artificial intelligence.

Dave leads the knowledge representation and natural language group at Thinking Machines. Product development efforts are focused on software that accepts unformatted English language data bases and provides convenient access and ultimately question-answering abilities for the data bases.

Allen M. Waxman
Senior Scientist

Ph.D., University of Chicago.

Prior to joining Thinking Machines, Allen was Associate Research Scientist at the Computer Vision Laboratory of the University of Maryland. He has also been a Visiting Research Fellow at the Weizmann Institute of Science and Instructor of Applied Mathematics at MIT. While at Maryland, Allen developed Image Flow Theory for the recovery of 3D structure and motion of objects from time-varying imagery. He also designed and directed development of a visual navigation system for the DARPA Autonomous Land Vehicle project.

Allen manages the machine vision group at Thinking Machines. Current research and product directions include systems for satellite image analysis, navigation, and autonomous docking.

Corporate Fellows

Richard P. Feynman
Richard Chace Tolman Professor of Theoretical Physics
Caltech

Ph.D., Princeton University.

During his 35 years as a Professor at Caltech, Richard has received the highest honors in his field. He received the Albert Einstein Award from Princeton in 1954 and The A.E.C./E.O Lawrence Award in 1962. He was elected a Foreign Member of the Royal Society in 1965, and received the Nobel Prize in Physics the same year. He received the Oersted Medal for Teaching in 1972 and the Niels Bohr International Gold Medal in 1973.

Numerical algorithms and methods of computation have been specialties of Richard's throughout his career. He has been in the forefront of Thinking Machines' work in developing new algorithms for computing in parallel. His analysis of message traffic and routing algorithms played a central role in the final design of the Connection Machine system.

Charles Leiserson
Professor
MIT

Ph.D., Carnegie-Mellon University.

As a graduate student at Carnegie-Mellon, Charles wrote the first paper on systolic architectures with H.T. Kung, for which they received a U.S. patent. His dissertation, *Area-Efficient VLSI Computation*, won the first ACM Doctoral Dissertation Award. In 1981, he joined the faculty of the theory of computation group in the MIT Laboratory for Computer Science. Honored by a Presidential Young Investigator Award in 1985, he is also a member of the IEEE and the ACM, and serves on the ACM General Technical Achievement Award Committee which selects the Turing Award winner. He has authored over twenty papers on the theory of VLSI and parallel algorithms.

At Thinking Machines, Charles applies his expertise on parallel computation, VLSI architectures, graph theory, digital circuit timing, analysis of algorithms, computer-aided design, placement and routing, wafer-scale integration and layout compaction, to the Connection Machine design and applications effort.

Tomaso A. Poggio
Professor
MIT

Doctor in Physics, University of Genoa.

Tomaso is Professor at the Artificial Intelligence Laboratory and at the Department of Psychology at MIT, with a joint appointment in the Whitaker College of Health Sciences and Technology. He has published widely in the field of vision and is on the Editorial Board of several specialized journals. He delivered the Second Annual David Marr Memorial Lecture at Cambridge University in 1983.

Tomaso plays a leadership role in the vision research now underway at Thinking Machines. Among his special interests is stereo vision, a capability which is being applied to the problems of satellite image analysis.

Jacob T. Schwartz
Professor
New York University

Ph.D., Yale University.

Jack is Professor of Mathematics and Computer Science at New York University and former Chairman of the Computer Science Department at the Courant Institute of Mathematical Sciences. He is a member of the National Academy of Sciences, the Editorial Board Chairman of the *Journal of Computer & Systems Sciences*, and the former Chairman of the Computer Science Board - National Research Council. He currently directs the New York University Robotics and Computer Vision Laboratory.

As a founder of the Ultracomputer project at the Courant Institute, Jack is a leading figure in parallel system design. At Thinking Machines he is actively involved in algorithms for massively parallel computation.

Jerome Wiesner
President Emeritus and Institute Professor
MIT

Ph.D., University of Michigan.

Jerome Wiesner is currently teaching and involved with research at MIT. He has been a member of the MIT Faculty for over 25 years, having served as Dean of the School of Science and Provost before becoming President in 1971. He was Science Advisor to Presidents Kennedy and Johnson. Jerome serves on many corporate boards and is an advisor to numerous national and international agencies.

At Thinking Machines, Jerome's role is that of scientific advisor for the company's basic and applied research projects.

Stephen Wolfram
Institute for Advanced Study
Princeton

Ph.D., Caltech.

An internationally known theoretical scientist, Stephen has contributed to a wide range of scientific and computational fields. He is the acknowledged leader in the burgeoning field of cellular automata, a field in which he has published widely. Among his many honors is the MacArthur Foundation Prize.

The Connection Machine system is the first computer system that can fully exploit the computational power of cellular automata. Stephen's work at Thinking Machines has centered on the use of these algorithms for the simulation of physical phenomena such as fluid dynamics and heat transfer.

Board of Directors

Richard J. Clayton

James Cohen
Kassko Enterprises

Louis Davis
Chairman, Churchill International

Marvin Denicoff

Sheryl Handler

W. Daniel Hillis

William McCowen
Hoenig & Co., Inc.

William S. Paley
Founder Chairman, CBS Inc.
Chairman, Executive Committee, CBS Inc.

Howard L. Resnikoff

Frank Stanton
President Emeritus, CBS Inc.
Former Chairman, The Rand Corporation.

™Connection Machine is a trademark of Thinking Machines Corporation.
™C* and CM Lisp are trademarks of Thinking Machines Corporation.
© 1985 Thinking Machines Corporation.

THE CONNECTION MACHINE™ COMPUTER
A Natural Fit To Applications Needs

The Connection Machine computer is a massively parallel system which reconfigures itself to match the natural structure of applications exactly, while providing computing power in excess of 1000 MIPS. Unique among parallel architectures, the system allows the connections among its processors to be altered at will. It is the next major advance in large scale computing.

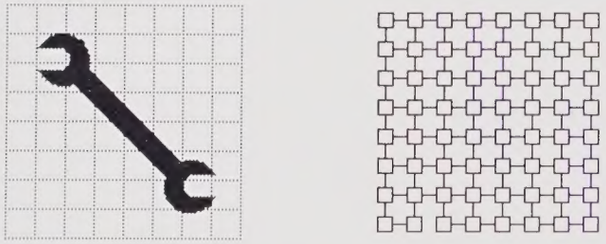

Figure 1:
The Connection Machine system configured for image processing.

The Connection Machine system is unique in its ability to adapt to the natural structure of the problem it is addressing. There are three steps in setting up the system to match the target problem. The first is to assign a processor to each individual problem element. The second is to arrange the interconnections among the processors to match the way individual problem elements interact. The third is to execute the calculations on all these interconnected elements concurrently.

A Processor For Each Problem Element

All large scale computer systems have to deal with overwhelming numbers of problem elements. In science and engineering, it is the quest for detail that creates huge numbers of elements. A simulation of an airplane wing, for example, may involve the tracking of turbulence at 200,000 individual points around the wing surface. A weather simulation may operate on 250,000 points in the atmosphere. In artificial intelligence and knowledge representation applications, it is the number of alternatives that raise the complexity level. A system may be required to choose intelligently from among 10,000 hypotheses.

Traditional large scale systems have dealt with detail by sequentially cycling a small number of processors (1 to 4) through all the problem elements as quickly as possible. That is why they are called serial systems. The Connection Machine dynamic computer employs a more powerful and simpler way. It assigns a complete processor to each problem element, whether there are 50,000 of them or a million.

Matching the Way Problem Elements Interact

Individual elements of a problem do not exist in isolation. They interact and influence the state of other elements. It is, in fact, the way they interact that is at the core of the problem. In a VLSI simulation, for example, the fact that a transistor has changed state is only one part of the situation. The effect of this state change on the other transistors it is connected to is the really important element.

No two problems have the same internal structure. Each has its own characteristic interaction paths, or "topology." The Connection Machine system, unique among large scale systems, adapts itself accordingly. Some problems, like image processing, have very regular connections. Elements of the image are held in "pixels." Processing the image involves large numbers of interactions between neighboring pixels. Therefore, for this application, the system organizes itself as a grid. Each processor is connected to the processor above, below, and to the sides.

VLSI simulation is a very different type of problem. The connection arrangement represents the wiring of the circuit; it is unique to an individual chip. A transistor may be connected to the one next door, the one clear across the chip, or both. Again, the Connection Machine system organizes itself accordingly. A processor is assigned to each transistor. A connection is established for each wire.

Fast Fourier Transform (FFT) applications also have a distinctive topology, which is matched exactly by the system. The Connection Machine system is very efficient at FFT's. Artificial intelligence, natural language processing, and knowledge representation are still a fourth important class of problems with a distinctive topology. Here problem elements link together in a variety of tree or graph structures. Each branch point can, for example, represent an alternative decision.

Connection Machine processors are not pre-wired in any of these topologies. Each of them is accommodated with equal ease, because all linkages are software controlled. Two processors that know each other's addresses are thereby linked together. System-wide message flow is handled by a 3 gigabit per second message routing system.

Computing in Parallel

All giant, high performance computers, parallel or serial, contain about a square meter of silicon circuitry. The difference is in how that square meter is utilized. A serial computer has a single central processor which uses about 5% of the

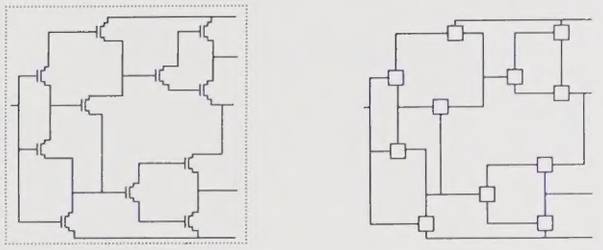

Figure 2:
The Connection Machine system configured for VLSI simulation.

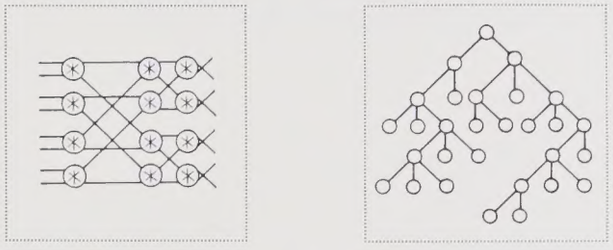

Figure 3:
FFT's and Trees : two more topologies that are matched exactly by the Connection Machine system.

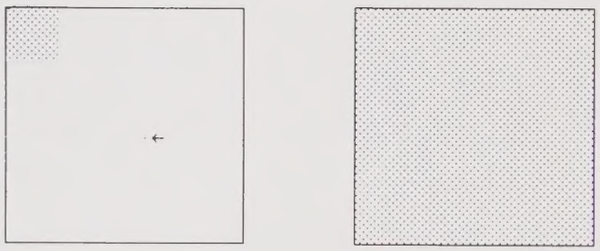

Figure 4:
(a) A serial architecture has a busy cpu, but the rest of the silicon, the memory, is hardly busy at all.
(b) The Connection Machine system has massive numbers of processors and hence keeps more of its circuitry busy.

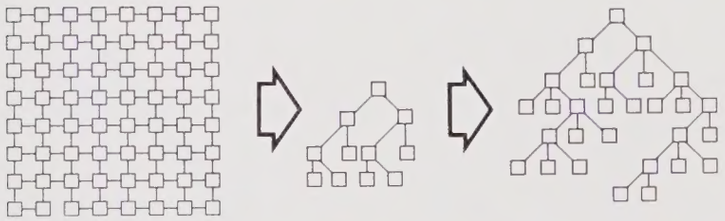

Figure 5:
A machine vision application requires a grid structure for initial pixel analysis, then a series of tree structures as the system works to identify the objects detected. The changes are made dynamically, simply by changing addresses in individual processors.

silicon area. The rest is used for memory.

This serial architecture, the von Neumann architecture, originated well before the availability of solid-state memory. When central processing modules and memory arrays were made of completely different components, there was no easy way to trade them off or make them physically close. The most practical approach was to house the processing logic and the memory separately, connected by a memory bus. The consequence is that the memory as a whole is not kept busy at all; only one location is active at a time. This limitation is known as the von Neumann bottleneck.

Recognizing that processors and memory are now made from the same silicon, the Connection Machine system's designers included huge numbers of processors. Furthermore, they distributed these processors throughout the memory. In this way tens of thousands of memory locations are busy at a time, not just one. By keeping more of its square meter of circuitry busy all the time, the Connection Machine system outperforms serial systems by a wide margin and is more cost-effective as well.

The one-to-one match of processors and problem elements in the Connection Machine system is implemented at two distinct levels. First, there are the 65,536 physical processors in the machine. Each is a complete computer with its own memory. All 65,536 computers fetch data and compute in parallel. At the second level of implementation, individual physical processors are partitioned by software into virtual processors. The enormous total horsepower of the system makes virtual processing feasible for problems with as many as a million elements.

Dynamic Reconfiguration

The Connection Machine reconfiguration capability is especially valuable for problems whose structure changes dynamically. New processor connections may be established at any time, simply by storing the new addresses. A single Connection Machine system can often replace multiple special-purpose machines, each of which can only handle one stage

of the problem.

Using the Connection Machine System

The Connection Machine system is not complicated to use. Both the hardware and the software employ a small set of very powerful tools. The speed of the system comes from the way these tools are used.

Access to all Connection Machine capabilities is via a front-end processor, either a Symbolics 3600[1] or a Digital VAX[2]. This front end provides the operating system environment, including terminal interaction and file management. It also holds the applications programs. Applications may be written in either C*™ (a Connection Machine extension of C) or CM Lisp™ (a Connection Machine extension of Lisp.)

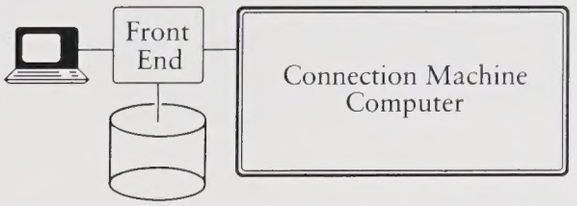

Figure 6:
The Connection Machine Front End

The first step in solving a large-scale problem is to break it down into its individual problem elements and assign one element to a processor. If there are 1,000,000 such problem elements, the machine is configured for a million virtual processors. The command to do this is contained in the applications program which resides in the host machine. It is transferred to the Connection Machine hardware for execution.

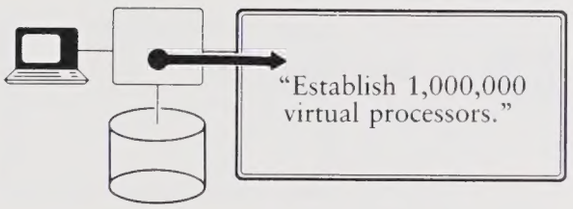

Figure 7:
Setting up the Virtual Processors.

The next step is to initialize each virtual processor to the state it will be in when execution begins. Each problem element will have a set of numbers that characterizes its initial state. It will also have a list of the other problem elements that it interacts with. This list will be in the form of pointers to the processors which contain these problem elements. If the problem is a VLSI circuit simulation, each element is a transistor. The state information indicates whether that transistor is on or off. The pointers tell which other transistors are wired to the output of this one.

Each virtual processor in the system has its own memory. When the Connection Machine hardware is operating as a physical machine with 65,536 processors, each processor has 4096 bits of memory, for a system total of 32 Mbytes. If the system is set up for a greater number of virtual proces-

[1]"Symbolics 3600" is a trademark of Symbolics, Inc.
[2]"VAX" is a trademark of Digital Equipment Corp.

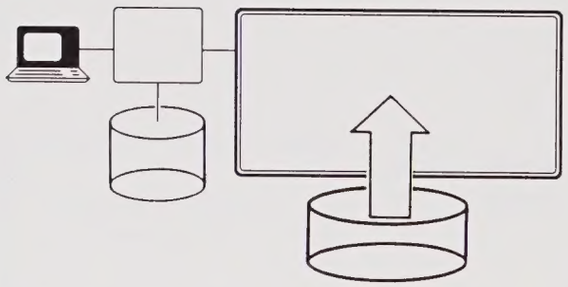

Figure 8:
Loading Data into the Connection Machine Memory.

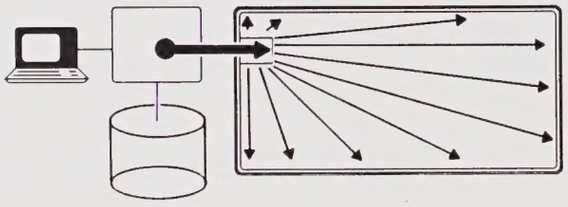

Figure 9:
Parallel Execution of Instructions.

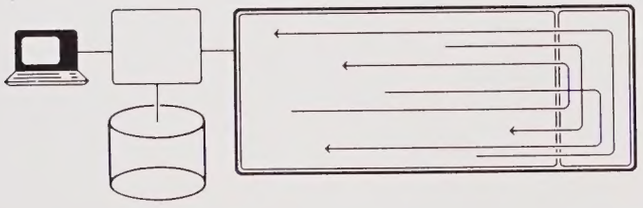

Figure 10:
A Message Communication Cycle Using the Router.

sors, the memory per processor is correspondingly smaller.

Loading of the initial state information proceeds in one of two ways. If the amount of data is small, it is loaded directly from the front end. All 32 Mbytes of Connection Machine memory are dual-ported and hence directly accessible by both machines. If the amount of data is large, such as the initialization data for a million-element simulation, it is loaded directly from the system's swapping disk. This 1.2-gigabyte disk system transfers to and from the Connection Machine memory at the rate of 500 megabits per second.

Once the problem elements are set up, program execution proceeds under control of the front end. The program, stored on the front end's disk, is loaded and executed. If the instructions are serial (i.e., they only need to be executed once) they are handled directly by the front end processor. Parallel instructions are passed to the Connection Machine hardware for execution. A microcontroller receives the instructions, expands them into Connection Machine "nano-instructions" and broadcasts these to all processors in the system.

Parallel instructions are of two types. The first type computes: manipulating information within each virtual processor. In a simulation, the computation would update the state of each problem element. The second type of instruction communicates: passing information from one problem element to another. Communication is a parallel instruction; all processors may send messages in parallel. An extremely high speed pipelined router manages message delivery.

From an applications programming perspective, this is how the Connection Machine system operates. It is simple to use because it fits so naturally with the problem to be solved. But backing up that simplicity is 1000 MIPS of computing power.

Summarizing the Connection Machine System

Every element of the Connection Machine system contributes to its applications orientation. Everything focuses on making large-scale problem solving easier. Dynamic reconfiguration sets the computer up the same way the problem is set up. Virtual processors allow a 1-on-1 matching of problem elements and processors. 65,000 physical processors provide the raw computing power (in excess of 1000 MIPS) to solve giant problems quickly. The speed of the router (over 3 billion bits per second) accommodates problems with massive amounts of intercommunication. And yet all this power is delivered with just a handful of new constructs in CM Lisp and C*.

Thinking Machines Corporation believes that advances in large scale computing require both power and simplicity. The power gives access to the needed detail. The simplicity allows the user to forget about the components of the computer and focus on the components of the problem. The magic of the Connection Machine system is that it provides both.

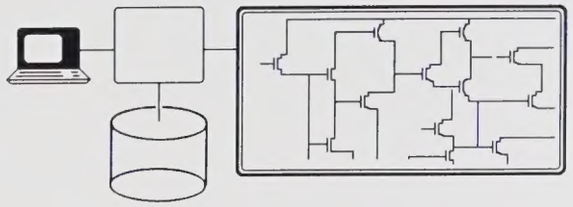

Figure 11:
The Connection Machine system is easily set up for VLSI simulation.

™Connection Machine is a trademark of Thinking Machines Corporation.
™C* and CM Lisp are trademarks of Thinking Machines Corporation.
© 1985 Thinking Machines Corporation.

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts
02142-1214

(617) 876-1111

The Connection Machine™ Computer

A More Detailed Look

THE CONNECTION MACHINE™ COMPUTER
A More Detailed Look

The Connection Machine system operates by receiving a stream of instructions from the front end, expanding each of these into a series of machine instructions, then broadcasting these machine instructions, one at a time, to all processors in parallel. The instructions coming in from the front end are referred to as "macro-instructions." The instructions broadcast to the individual processors are called "nano-instructions."

The Connection Machine system has 65,536 physical processors, but may be configured for a much larger number of logical processors by means of the *global-reset* and *configure* commands. *Configure* takes two arguments, allowing a two dimensional array of virtual processors per physical processor. *Configure 4,4*, for example, sets up the machine in the million-processor mode. (Or, more precisely, the 1,048,576 processor mode.) The same number of virtual processors could be established by the command *configure 16,1*. Since virtual processors are so commonly used on the system, they will be referred to simply as "processors." Where it is necessary to refer to one of the 65,536 hardware processors, the term "physical processor" will be used.

Each physical processor has 4096 bits of memory, making 32 Mbytes for the machine as a whole. In the million-processor mode, each processor has 256 bits of memory. Memory is divided into a data area and a stack area, with the layout being the same in each processor. A single, system-wide register, the stack limit, defines the boundary between stack space and data space. The stack pointer is also a system-wide register. The stacks in all processors act in unison.

Memory is bit-addressable; fields can be of arbitrary length. There are three standard numeric formats: unsigned integer, signed integer, and floating point. Each is of arbitrary length. In particular, floating point numbers can be of any length. (If they are 32 bits or 64 bits long, they conform to the IEEE standard.)

Addresses in the system as a whole operate at three levels. The first is the physical processor level. Corresponding to

each physical processor, there are one or more virtual processors. Corresponding to each virtual processor, there are a number of bits of memory.

Data may be exchanged between the Connection Machine memory and the front end in any of three ways. *Read-slice* reads a single bit of information from the memory of each of a series of consecutive processors, assembles them into a signed integer, and passes the integer to the front end. *Write-slice* goes the other way. Slice operations are typically done 16 or 32 processors at a time. *Read-processor* and *write-processor* move a single field between the front end and a single processor. *Read-array* and *write-array* move fields between the front end and a set of contiguous processors.

All instructions flow into the Connection Machine hardware from the front end. These macro-instructions come into a microcontroller, which expands them into a series of nano-instructions. Some expand into just a few nano-instructions. Others expand into as many as a hundred or more. It is also possible to feed nano-level instructions to the microcontroller and control the hardware directly. It is not, however, efficient to do so, because the front-end cannot supply these instructions rapidly enough to keep the system busy. (Direct control of the hardware from the front end is provided primarily so that the front end can support debugging and diagnostic aids.)

Nano-instructions are broadcast to all virtual processors. Processors, however, have the option of "sitting out" a series of instructions. A one-bit flag within each processor, the Context Flag, determines whether that individual processor will respond to the instruction or not. Most of the instructions discussed below are "conditional" in the sense that they only take effect in the processors whose Context Flag is 1.

The Connection Machine system is implemented with four physical microcontrollers, one for each 16,384 processors. If the system has a single front end, that front end is connected

to all four microcontrollers and therefore drives all 65,536 processors in parallel. A system may be configured with up to four front ends. A switch called the Nexus makes the connections between front ends and microcontrollers. It is possible, for example, for four users to be operating simultaneously. Each works at a separate front end, and each has a separate instruction stream executing in a quarter of the system's processors. For the remainder of this document, it is assumed that the system is operating with a single front end.

Processor Instructions

The system's computational macro-instructions will be familiar to the user of any modern macro assembler. They operate on signed integers, unsigned integers, and floating point values. They include unary operators such as *not*, *negate*, and *absolute-value*. They include binary operators such as *and*, *or*, *add*, *subtract*, *multiply*, *divide*, *compare*, *shift*, and *move*. What is unique about these instructions on the Connection Machine computer is that they operate on every selected processor in parallel. The results, of course, are different in each processor, because each is doing the operation on its own data.

In the case of the *random* instruction, this difference is explicit. *Random* places an independently chosen random number in each selected processor. Two processors may or may not be assigned the same random value. The *enumerate* operation, on the other hand, guarantees to assign a distinct integer to each selected processor.

Global instructions produce a single result from data items stored in the memories of all selected processors. *Global-logical-or* for example, takes the inclusive or of a field in each processor's memory. *Global-count* examines a single-bit field in all processors and returns the number of "1" bits. *Global-add* sums multi-bit fields. *Global-max (-min)* returns the largest (smallest) value found in a specified field across all selected processors. *Global-add* and *global-count* operate on unsigned integers, signed integers, and floating point values, as does *global-max (-min.)*. The *enumerate* instruction

goes in the other direction, placing a different consecutive integer into each of a selected range of processors.

A Connection Machine processor is a single-bit computer which operates on two data values, A and B. Each processor also has an array of flags which can participate in operations. Instructions specify the address of A (within the processor's own memory), the address of B, and the number of the flag to be used. Results are always stored back into A. A flag value may be stored as well. Since an instruction at this level involves three input bits, there are 256 possible Boolean operations. All 256 are implemented.

The simplest form of communication between Connection Machine processors is between nearest neighbors. Each processor is wired to its neighbors to the North, East, West, and South by a communications grid called the NEWS network. Four instructions, *get-from-north*, *get-from-east*, *get-from-west*, and *get-from-south* control the transfer of data. NEWS information is passed one bit at a time.

Beside the NEWS grid is a much more powerful communications system, the Connection Machine Router. It allows full messages to be sent from any processor to any other; the sending processor simply needs to know the address of the destination processor. Messages may be of any length. Typical messages contain 32 bits of information; adding the address information and headers results in a transmitted package of 52 bits.

Each of the 65,536 physical processors is connected to 16 other physical processors in a special organization called a "Boolean n-cube." Because 65,536 is expressed in 16 bits in binary, the Connection Machine's cube is a Boolean 16-cube. Each of the processors is connected to every other processor whose binary address is different in just one of the 16 bits. The following example shows the interconnections of processors 6_{10} and 2070_{10}. The binary addresses are shown in parentheses.

These two sets of addresses have a common connection. Processors 6 and 2070 both connect to 22. Thus it is possi-

2	(0000 0000 0000 0010)
4	(0000 0000 0000 0100)
6	(0000 0000 0000 0110)
7	(0000 0000 0000 0111)
14	(0000 0000 0000 1110)
22	(0000 0000 0001 0110)
38	(0000 0000 0010 0110)
70	(0000 0000 0100 0110)
134	(0000 0000 1000 0110)
262	(0000 0001 0000 0110)
518	(0000 0010 0000 0110)
1030	(0000 0100 0000 0110)
2054	(0000 1000 0000 0110)
4102	(0001 0000 0000 0110)
8198	(0010 0000 0000 0110)
16390	(0100 0000 0000 0110)
32774	(1000 0000 0000 0110)

Figure 1:
The sixteen processors that are connected to processor 6_{10}.

22	(0000 0000 0001 0110)
2054	(0000 1000 0000 0110)
2066	(0000 1000 0001 0010)
2068	(0000 1000 0001 0100)
2070	(0000 1000 0001 0110)
2071	(0000 1000 0001 0111)
2078	(0000 1000 0001 1110)
2102	(0000 1000 0011 0110)
2134	(0000 1000 0101 0110)
2198	(0000 1000 1001 0110)
2326	(0000 1001 0001 0110)
2582	(0000 1010 0001 0110)
3094	(0000 1100 0001 0110)
6166	(0001 1000 0001 0110)
10262	(0010 1000 0001 0110)
18454	(0100 1000 0001 0110)
34838	(1000 1000 0001 0110)

Figure 2:
The sixteen processors that are connected to processor 2070_{10}.

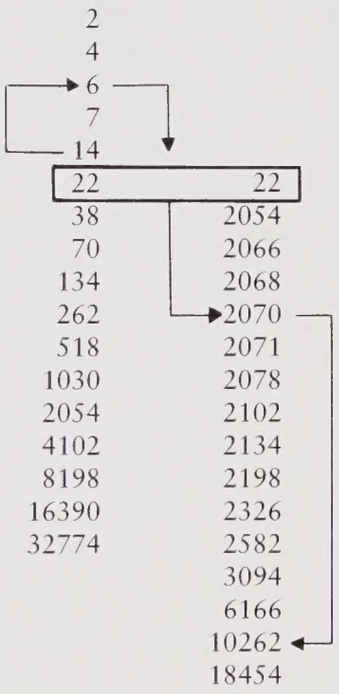

Figure 3:
A four step routing processor.

From Processor	With Relative Address	To Processor	
14	(0010 1000 0001 1000)	6	(move −8)
6	(0010 1000 0001 0000)	22	(move +16)
22	(0010 1000 0000 0000)	2070	(move +2048)
2070	(0010 0000 0000 0000)	10262	(move +8192)
10262	(0000 0000 0000 0000)	---	(done)

Figure 4:
Using relative addresses to compute the routing path.

ble to pass a message, for example, from processor 14 to processor 10262 in just four steps. The router at processor 14 passes it to the router at processor 6, which passes it to 22. From there it goes to 2070 and then to 10262.

A four step routing process

The illustration above was worked out using absolute processor addresses. In fact, the router uses relative addresses. Instead of saying, "Send this message to processor 10262" processor 14 actually says, "Send this message to the processor whose relative address is 10264." This relative address is obtained by XOR'ing the source and destination addresses. The XOR of 0000 0000 0000 1110 and 0010 1000 0001 0110 is 0010 1000 0001 1000. This relative address does more than just indicate the destination. It defines the set of possible routes. Each "1" bit marks a required routing step. The right-most "1" bit is in the bit position whose value is 8_{10}. This bit tells the router to route the message through the chip which is "down 8," in other words, processor 6. At the same time the router sends the message, it clears this bit, so that processor 6 receives a message that says "Send this message to the processor whose relative address is 0010 1000 0001 0000." The destination processor is still 10262. The pattern continues until the message arrives. At this point the relative address is zero, indicating that no more moves are needed.

The routing steps indicated by a relative address may be done in any order. In the simplest case, the 1-bits indicating the required routine steps for a message are processed in right-to-left order. If there is too much traffic along a particular path, however, some messages will take their routing steps in a different order, thus guiding them along alternate paths.

Connection Machine physical processors are grouped sixteen to a chip. There is a single router on each chip that services all sixteen processors. Hence four of the sixteen routing connections are internal to an individual chip. As a result, it takes a maximum of twelve steps to move from any

chip to any other chip. During message routing, the system goes through all twelve steps. If the router on a given chip has a message whose relative address has a "1" in the low order bit position, it sends that message on the first of the twelve steps to the chip whose address differs in that same bit (i.e., the next chip). If the message it contains has a "0" in the low order relative address bit, it doesn't send anything on that step. The process continues through all twelve steps, with all router chips responding in the same way.

The basic message passing instruction is *store*. *Store* specifies the length of the message as well as the relative address of the destination processor. *Store* is a parallel instruction; all selected processors initiate message transfers at once. Special routing hardware handles the volume of messages efficiently. An individual router on a chip may receive as many as 12 messages from other chips during a message cycle, one from each other chip that it is connected to. It can in turn send as many as 12 messages, one on each of the wires. If two messages need to go down the same wire, one is buffered until the next routing cycle. Each of the 4096 routers can buffer as many as 7 messages. If an individual router becomes extremely busy, it can defer acceptance of any new messages from its own processors. This deferral keeps the router free to handle messages from other chips. If the chip's buffer space still fills, it refers messages to neighboring chips.

Parallel message sending also introduces the possibility that the same location in the same processor will receive two or more messages in the same cycle. The simple *store* instruction gives unpredictable results in this case. Several variations of the *store* instruction, such as *store-with-add*, deal with this possibility. If two or more *store-with-add* messages arrive at the same destination, they are summed.

Router throughput has been measured at 3 gigabits per second under heavy load. The test routine repeatedly sent 32 bit messages from all 65,536 processors in parallel. Destination processors were chosen at random, but each message always had a unique destination.

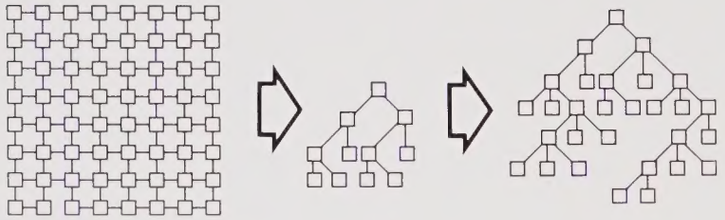

Figure 5:
A machine vision application requires a grid structure for initial pixel analysis, then a series of tree structures as the system works to identify the objects detected. The changes are made dynamically, simply by changing addresses in individual processors.

Dynamic Reconfiguration

A processor address is all it takes to establish a linkage on the system. This flexibility allows applications to reconfigure on the fly. A number of instructions support this capability. The *my-address* instruction allows processors to determine their own addresses, so they can send them to other processors and thus establish new connections. The *processor-cons* instruction allows each selected processor to find another "free" processor.

Processor-cons specifies the address of a one-bit field, the "free flag." A processor is considered free if it has a "1" in that field. The system looks in parallel for processors with 1's and passes the address of one to each selected processor, clearing the bit at the same time.

A machine vision application requires a grid structure for initial pixel analysis, then a series of tree structures as the system works to identify the objects detected. The changes are made dynamically, simply by changing addresses in individual processors.

Programming the Connection Machine

The Connection Machine programming languages deliver the power of parallel processing in a natural and familiar way. REL-2, CM Lisp™, and C*™ will be instantly familiar to assembly language programmers, Lisp programmers, and C programmers respectively. In each case a very compact set of parallel operators is added to access the massively parallel architecture. The REL-2 and CM Lisp languages are well-defined and are discussed below. Details of the C* language extensions will be released by Thinking Machines Corporation as soon as they are available.

REL-2 is an assembly level language for the system. Its primary role is as a language in which to develop CM Lisp and C*, but it is available for independent use. The REL-2 user interface on the front end consists of a set of Lisp macros and functions to be called from Lisp code. These macros

and functions then direct the actions of the system by sending directives to the Connection Machine microcontroller. The previous section of this document gives the flavor of REL-2. All the italicized commands used as examples in that section are REL-2 commands.

CM Lisp will be immediately familiar to the Lisp programmer. Program flow and control are essentially the same as in a serial Lisp. Parallel operations utilize a data structure called a xector (pronounced zek′ tor.) Xectors allow values to be stored one per processor, allowing simultaneous operations on all the elements. A xector is defined by three components: a domain, a range, and a mapping between them. Each element in the domain is called an index and must be unique; each element in the range is called a value. Stored one per processor, the index functions as the name of the processor. In the xector, {sky→blue grass→green apple→red} sky, grass, and apple are the indices. Blue, green, and red are the values. If the domain of a xector is the sequence of integers starting with zero, the xector may be written in a more compact form. {0→A 1→B 2→C 3→ D} is equivalent to [A B C D].

Because the Connection Machine system operates in parallel on xectors, many of the generic Lisp sequence operations speed up dramatically when used with them. Operations such as SEARCH and DELETE, which can be performed on each element independently, execute in a fixed time no matter how many elements are in the xector. Operations which involve reducing, counting, or numbering the elements take place in logarithmic time, because they are implemented by algorithms on balanced trees.

The Greek letter α is used by CM Lisp to indicate that an operation is to be done many times in parallel. One informal way to think of α is that it means "give me a zillion" of whatever is inside the expression, where "a zillion" is however many are needed. The α notation will produce a zillion additions, a zillion threes, or whatever. For example, (α + {a→1 b→2} {a→3 b→3}) evaluates to {a→4 b→5}. In this case "a zillion times" means twice, because there are just

two elements in the xectors. The α may be factored out of the expression for clarity. Thus α(+ 1 2) is equivalent to (α+. α1 α2).

The Greek letter β is used to define a second kind of parallel operation. β converts a two-argument function into a function that reduces the elements of a xector into a single value. This reduction is performed in logarithmic time. β reduction uses only the values of the element and ignores the indices. For example, (β + {A→1 B→2 C→3}) evaluates to 6. (βMAX [1 3 5 7]) evaluates to 7. An extended form of β can perform many such reductions in parallel.

These are the primary constructs that have been added to the CM Lisp language. In terms of the hardware, α broadcasts single values and performs parallel independent computations; β gathers information together and performs general interprocessor communication.

Summarizing the Connection Machine System

Every element of the Connection Machine system contributes to its applications orientation. Everything focuses on making large-scale problem solving easier. Dynamic reconfiguration sets the computer up the same way the problem is set up. Virtual processors allow a 1-on-1 matching of problem elements and processors. 65,000 physical processors provide the raw computing power (in excess of 1000 MIPS) to solve giant problems quickly. The speed of the router (over 3 billion bits per second) accommodates problems with massive amounts of intercommunication. And yet all this power is delivered with just a handful of new constructs in CM Lisp and C*.

Thinking Machines Corporation believes that advances in large scale computing require both power and simplicity. The power gives access to the needed detail. The simplicity allows the user to forget about the components of the computer and focus on the components of the problem. The magic of the Connection Machine system is that it provides both.

™Connection Machine is a trademark of Thinking Machines Corporation.
™C* and CM Lisp are trademarks of Thinking Machines Corporation.
© 1985 Thinking Machines Corporation.

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts
02142-1214

(617) 876-1111